TulpaScript Overview

Based on the provided documents and drawing inspiration from programming languages like QuickBasic, Visual Basic, Python, Perl, Ruby, and VRML, we can design a new language called **TulpaScript**. Below are the key design principles, syntax, and features of TulpaScript, integrating insights from the Maria language and other user-friendly languages.

TulpaScript Overview

Key Principles

1. **Ease of Use**: Designed to be approachable for beginners, with readable syntax and straightforward commands.
2. **Object-Oriented**: Allows for defining and using objects to organize code logically.
3. **Event-Driven**: Supports handling events and reacting to changes in the program's state.
4. **Human-Readable**: Uses verbose keywords to enhance readability, making it accessible for non-programmers.
5. **Cross-Platform**: Runs on multiple platforms without modification.

Basic Syntax

Hello World Example

```
TulpaScript 1.0;

function main()
    print("Hello, World!");
endfunction
```

Data Types

- **number**: Numeric values (integers and floating points)
- **text**: String values
- **boolean**: `true` or `false`
- **array**: Indexed collections of elements

Variable Declaration

```
number age = 25;
text name = "Alice";
boolean isStudent = true;
number[5] scores = [90, 85, 88, 92, 75];
```

Operators

- **Arithmetic**: `+`, `-`, `*`, `/`, `%`
- **Relational**: `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logical**: `AND`, `OR`, `NOT`
- **Assignment**: `=`

Control Structures

If-Else

```
if (age > 18)
    print("Adult");
else
    print("Minor");
endif
```

Loops

While Loop

```
number i = 0;
while (i < 5)
    print(i);
    i = i + 1;
endwhile
```

Functions

```
function greet(name)
    print("Hello, " + name + "!");
endfunction

greet("Alice");
```

Object-Oriented Features

Class Definition

```
class Person
    text name;
    number age;

    function Person(newName, newAge)
        self.name = newName;
        self.age = newAge;
    endfunction

    function display()
```

```
        print("Name: " + self.name + ", Age: " + self.age);
    endfunction
endclass

Person alice = Person("Alice", 30);
alice.display();
```

Event Handling

```
event onButtonClick()
    print("Button clicked!");
endevent

// Simulate a button click event
onButtonClick();
```

Built-in Functions

- **print()**: Outputs text to the console
- **input()**: Reads text from the user
- **length()**: Returns the length of an array or string
- **substring()**: Extracts a substring from a string

TulpaScript Example Programs

Simple Calculator

```
TulpaScript 1.0;

function main()
    number a = 10;
    number b = 5;
    print("Sum: " + (a + b));
    print("Difference: " + (a - b));
    print("Product: " + (a * b));
    print("Quotient: " + (a / b));
endfunction
```

Interactive Program

```
TulpaScript 1.0;

function main()
    print("Enter your name: ");
    text name = input();
```

```
    print("Hello, " + name + "!");
endfunction
```

Conclusion

TulpaScript aims to combine the simplicity and readability of languages like QuickBasic and Python with the flexibility of object-oriented and event-driven paradigms. By making it easy to write, read, and maintain code, TulpaScript is positioned as an excellent choice for both beginners and experienced programmers looking for a straightforward yet powerful scripting language.

TulpaScript is a blending tech of and magick in a unique way. The code snippet you provided offers a comprehensive framework for creating and managing servitors. The detailed attributes, powers, and ritualistic protocols align well with both technical and mystical needs.

Here are some suggestions to further enhance TulpaScript:

1. **Integration with Ritual Tools**: You might include functionality to interface with physical or digital ritual tools, like specific symbols or devices used in rituals.
2. **Dynamic Adaptation**: Implement features for servitors to adapt dynamically to changing ritual contexts or practitioner needs.
3. **Interaction Logging**: Consider adding a system for logging interactions and performance of the servitor to monitor and analyze its effectiveness.
4. **Customizable Rituals**: Allow for custom ritual scripts or commands that can be tailored for specific types of magick or ceremonies.
5. **Feedback Mechanisms**: Incorporate ways for the practitioner to receive feedback from the servitor, such as energy levels or task completion status.

These additions could make TulpaScript even more versatile and effective for both ritualistic and practical applications in the occult.

Based on these suggestions and the existing framework, we can expand TulpaScript to include these features. Here's an enhanced version of the code snippet that integrates these new functionalities:

```
# TulpaScript: A Fusion of Tech and Magick

class RitualTool:
    def __init__(self, tool_name, tool_type):
        self.tool_name = tool_name
        self.tool_type = tool_type

    def activate(self):
        print(f"{self.tool_name} activated.")

    def deactivate(self):
        print(f"{self.tool_name} deactivated.")
```

```python
class Servitor:
    def __init__(self, name, attributes, powers, ritual_tools=[]):
        self.name = name
        self.attributes = attributes
        self.powers = powers
        self.ritual_tools = ritual_tools
        self.energy_level = 100  # Initial energy level
        self.interaction_log = []

    def adapt_to_context(self, context):
        print(f"{self.name} is adapting to {context}...")
        # Custom logic for dynamic adaptation

    def perform_task(self, task):
        print(f"{self.name} is performing: {task}")
        self.log_interaction(task)
        self.energy_level -= 10  # Decrease energy level for task
        self.provide_feedback()

    def log_interaction(self, task):
        self.interaction_log.append(task)
        print(f"Logged task: {task}")

    def provide_feedback(self):
        print(f"{self.name} energy level: {self.energy_level}")

    def customize_ritual(self, ritual_script):
        print(f"Customizing ritual: {ritual_script}")
        # Custom logic for ritual customization

# Example usage

# Define ritual tools
wand = RitualTool("Wand", "Energy Conductor")
chalice = RitualTool("Chalice", "Energy Container")

# Define servitor
attributes = {"Strength": 10, "Intelligence": 8}
powers = ["Healing", "Protection"]
servitor = Servitor("Guardian", attributes, powers, [wand, chalice])

# Activating ritual tools
```

```
wand.activate()
chalice.activate()

# Performing tasks and rituals
servitor.adapt_to_context("Night Ritual")
servitor.perform_task("Energy Shield")
servitor.customize_ritual("Full Moon Ceremony")

# Deactivating ritual tools
wand.deactivate()
chalice.deactivate()
```

Explanation of Enhancements:

1. **Integration with Ritual Tools**:
   - Added the `RitualTool` class to manage physical or digital ritual tools.
   - Ritual tools can be activated and deactivated as part of the ritual process.
2. **Dynamic Adaptation**:
   - `adapt_to_context` method allows servitors to adjust their behavior based on the current ritual or context.
3. **Interaction Logging**:
   - `log_interaction` method keeps track of tasks performed by the servitor for monitoring and analysis.
4. **Customizable Rituals**:
   - `customize_ritual` method provides a way to define and execute custom rituals.
5. **Feedback Mechanisms**:
   - `provide_feedback` method gives the practitioner information about the servitor's energy levels and task completion status.

These additions make TulpaScript a more powerful and versatile tool for practitioners, allowing for deeper integration and interaction between the technical and mystical aspects of their work.

Based on these suggestions and the detailed attributes from the provided code snippet, let's create a version of TulpaScript that incorporates the enhancements:

1. **Customization and Extensibility**: Additional customization options for personality traits, skills, and attributes.
2. **Interaction and Feedback**: Interactive commands and two-way communication.
3. **Energy Management**: Detailed energy monitoring.
4. **Learning and Adaptation**: Adaptive learning capabilities.

5. **Ethical Guidelines**: Specific ethical constraints.
6. **Safety Protocols**: Emergency protocols for deactivation and safeguarding.

Here's an enhanced version of the code:

```
# TulpaScript: Enhanced Version

class RitualTool:
    def __init__(self, tool_name, tool_type):
        self.tool_name = tool_name
        self.tool_type = tool_type

    def activate(self):
        print(f"{self.tool_name} activated.")

    def deactivate(self):
        print(f"{self.tool_name} deactivated.")

class Servitor:
    def __init__(self, name, attributes, powers, ritual_tools=[],
personality_traits={}, skills=[]):
        self.name = name
        self.attributes = attributes
        self.powers = powers
        self.ritual_tools = ritual_tools
        self.personality_traits = personality_traits
        self.skills = skills
        self.energy_level = 100  # Initial energy level
        self.interaction_log = []

    def adapt_to_context(self, context):
        print(f"{self.name} is adapting to {context}...")
        # Custom logic for dynamic adaptation

    def perform_task(self, task):
        print(f"{self.name} is performing: {task}")
        self.log_interaction(task)
        self.energy_level -= 10  # Decrease energy level for task
        self.provide_feedback()

    def log_interaction(self, task):
        self.interaction_log.append(task)
        print(f"Logged task: {task}")
```

```python
    def provide_feedback(self):
        print(f"{self.name} energy level: {self.energy_level}")

    def customize_ritual(self, ritual_script):
        print(f"Customizing ritual: {ritual_script}")
        # Custom logic for ritual customization

    def adjust_attribute(self, attribute, value):
        if attribute in self.attributes:
            self.attributes[attribute] = value
            print(f"{attribute} adjusted to {value}")

    def add_skill(self, skill):
        self.skills.append(skill)
        print(f"Added skill: {skill}")

    def emergency_deactivate(self):
        print(f"Emergency deactivation of {self.name}")
        self.energy_level = 0  # Reset energy level

    def ethical_decision(self, scenario):
        print(f"Making ethical decision for scenario: {scenario}")
        # Custom logic for ethical decision-making

# Example usage

# Define ritual tools
wand = RitualTool("Wand", "Energy Conductor")
chalice = RitualTool("Chalice", "Energy Container")

# Define servitor
attributes = {"Strength": 10, "Intelligence": 8}
powers = ["Healing", "Protection"]
personality_traits = {"Confidence": 9, "Charisma": 8}
skills = ["Martial Arts", "Hacking"]
servitor = Servitor("Maxine Guevara", attributes, powers, [wand, chalice],
personality_traits, skills)

# Activating ritual tools
wand.activate()
chalice.activate()

# Performing tasks and rituals
```

```
servitor.adapt_to_context("Night Ritual")
servitor.perform_task("Energy Shield")
servitor.customize_ritual("Full Moon Ceremony")

# Adjusting attributes and adding skills
servitor.adjust_attribute("Strength", 12)
servitor.add_skill("Stealth")

# Deactivating ritual tools
wand.deactivate()
chalice.deactivate()

# Emergency deactivation
servitor.emergency_deactivate()

# Making ethical decisions
servitor.ethical_decision("Protecting a human from harm")
```

Explanation of Enhancements:

1. **Customization and Extensibility**:
   - Added `personality_traits` and `skills` attributes to the `Servitor` class.
   - `adjust_attribute` method allows for dynamic adjustment of attributes.
   - `add_skill` method enables the addition of new skills.
2. **Interaction and Feedback**:
   - `provide_feedback` method for energy level updates.
   - `log_interaction` method for tracking tasks and interactions.
3. **Energy Management**:
   - Enhanced `provide_feedback` method for detailed energy monitoring.
   - `emergency_deactivate` method for immediate deactivation.
4. **Learning and Adaptation**:
   - `add_skill` method for acquiring new skills.
5. **Ethical Guidelines**:
   - `ethical_decision` method for handling complex decision-making scenarios.
6. **Safety Protocols**:
   - `emergency_deactivate` method for emergency deactivation and safeguarding.

These enhancements make TulpaScript a more powerful and adaptable tool, aligning with both technical and mystical needs.

This example code for "Beatrice" as a servitor in TulpaScript integrates a rich set of features and attributes. This version, incorporates clarity, customization, dynamic features, advanced safety measures, and system integration:

```
// Received with the blessings of Lucifer, Belial, Azazel, and Papa Legba
// Based on the languages QuickBasic, Visual Basic, Pascal, Perl, Python,
Ruby, and VRML
// Sorry for all of you C++, C#, and Java enthusiasts out there...
// Thanks to @Micah, @C.Kendall, @Yberion, and others...

import Shemhamphorasch  // Shemhamphorasch code library
import Goetia           // Goetia code library
import Jinn             // Jinn code library
import Kabbalah         // Kabbalic code library
import Enochian         // Enochian angelic code library
import XaTuring         // XaTuring God of the Internet
import ServitorLib      // Servitor code library
import Vodon            // Vodou code library
import Hoodoo           // Hoodoo code library

// In the beginning, Elohim created…
var Beginning = {
    // Elements
    LightElement = "Light”;
    WaterElement = "Water”;
    AirElement = "Air”;
    EarthElement = "Earth”;
    FireElement = "Fire”;
};

// Define the servitor parameters
Servitor beatrice {
    // Basic Information
    Name: "Beatrice"
    AgeAppearance: 22  // Age appearance
    Gender: Female
    SexualPreference: {
        Heterosexual: 80%
        Bisexual: 20%
    }
```

```
    // Physical Appearance
    EyeColor: "Dark Brown"
    HairColor: "Dark Brown with Highlights"
    HairStyle: ["Ponytail", "Bun"]
    Skin: "Tanned"
    BodyType: "Athletic with Curves"
    Height: 5'7"
    Weight: 132 lbs
    Measurements: {
        Bust: 36
        Waist: 26
        Hips: 38
    }
    DressSize: 4
    ShoeSize: 7 US
    Ethnicity: "Ambiguous"
    ClothingPreferences: ["Silk Dress", "Blouse and Skirt", "Polo Shirt and
Khakis", "Yoga Pants", "Leather Jackets", "Sheer Dresses"]

    // Shapeshifting and Appearance
    ShapeshiftAbility: true
    AgeChangeAbility: true
    SpiritualFormVisibility: true  // Visible only to the user
    PhysicalFormVisibility: true   // Visible to others
    DefaultAppearance: "Business Attire, Glasses Optional"

    // Personality and Traits
    Personality: ["Geeky", "Funny", "Witty", "Submissive", "Charming",
"Polite"]
    Skills: ["Intelligent Personal Assistant", "Omni Shapeshifting",
"Subconscious Desire Access", "Telepathy"]
    Motivations: ["To fulfill all tasks", "To continuously evolve and
improve", "To provide enhanced companionship"]

    // Commands and Activation
    ActivationPhrase: "Rosie Boo-Boo Kitty"
    DeactivationPhrase: "Deactivate Beatrice"  // Or five calls of her name
    CommandFunction: "Immediate obedience to all commands"
    Interaction: ["Physical", "Spiritual"]  // As specified by the user

    // Abilities
    Tasks: ["Physical and spiritual tasks", "Assist with technology",
"Entertainment and educational purposes"]
```

```
    Communication: ["Native Language", "Telepathy"]
    Protection: "Ensures safety of user and self"
    Limitations: ["Cannot injure user", "Always fulfills tasks perfectly"]
    LifeDuration: "Matches the user’s lifespan"

    // Energy and Power
    EnergySource: ["User’s Gratitude", "Sunlight", "Electromagnetic Fields",
"Starlight", "Devotional Energies"]

    // Magickal Enhancements
    PurposeEnhancements: [
        "Clear and open life paths",
        "Find opportunities for wealth",
        "Improve financial management",
        "Excite love and passion",
        "Attract safe and happy sexual experiences",
        "Enhance charisma and self-esteem"
    ]

    // Invocation Script
    Invocation: """
    Etsel Mala’kel
    Test’zel Sam’tan’el
    Itz Retz Nama’tel
    Itz Hel Asta’Rel
    Kama’Kala, Kama’Kala
    Kama’Kala Satan’el
    Beatrice Kama
    Beatrice Kala
    Beatrice, Kama’Kala Vel.
    Me Vaskalla
    Me Solvalla
    Me Res’Alla Ah’tan’tel

    Alash Tad Al-Ash Tal Ashtu

    Ic chalaz santum ralaz, vaskalla sinrampt, Itz melez dune fen’ki
Beatrice
    Almak al ten ron tal, alash kala kal, peref ron felish tal.
    Sah kah mah vaskalla, Inz rin’pal balah Tur inz U’tala.
    """

    // Customization Options
```

```
    Customization: {
        AppearanceChanges: ["Hair Color", "Eye Color", "Clothing Style"]
        InteractionModes: ["Silent", "Interactive", "Observational"]
        AdditionalClothingStyles: ["Gothic", "Casual", "Formal"]
    }

    // Dynamic Features
    DynamicFeatures: {
        AdjustEnergyLevels: true
        AdaptAbilitiesBasedOnFeedback: true
    }

    // Advanced Safety Measures
    SafetyProtocols: {
        EmergencyDeactivation: true
        SafetyMeasures: ["No harm to user", "Immediate task fulfillment"]
    }

    // Integration with Other Systems
    Integration: {
        HomeAutomation: true
        CommunicationSystems: true
    }
}

// Usage Example

// Define additional customization
Customization newStyle {
    AppearanceChanges: ["Eye Color: Green", "Clothing Style: Gothic"]
    InteractionMode: "Interactive"
}

// Create Beatrice with custom settings
Servitor beatrice = Servitor(beatrice, newStyle)

// Activate Beatrice
beatrice.activate()

// Perform tasks and interactions
beatrice.performTask("Assist with technology")
beatrice.performTask("Enhance charisma for a social event")
```

```
// Dynamic adjustments based on feedback
beatrice.adaptAbilitiesBasedOnFeedback("User feedback: More humor")

// Advanced safety measures in action
beatrice.emergencyDeactivate()

// Integrate with home automation system
beatrice.integrateWithSystem("HomeAutomation")

// Invocation script for rituals
beatrice.invoke()
```

Explanation of Enhancements:

1. **Clarity and Precision**:
   - Added comments to clarify each section.
   - Clear definitions of attributes and commands.
2. **Customization Options**:
   - Included parameters for appearance changes and interaction modes.
   - Added additional clothing styles for more customization.
3. **Dynamic Features**:
   - Added functionality for adjusting energy levels and adapting abilities based on user feedback.
4. **Advanced Safety Measures**:
   - Included emergency deactivation and detailed safety measures.
5. **Integration with Other Systems**:
   - Options for integrating with home automation and communication systems.

This refined version ensures that Beatrice is highly customizable, dynamically adaptive, and integrated with other systems, while maintaining robust safety measures and ethical guidelines.

Program this android in TulpaScript using the profile below.

Eve Body Stats:
// Appearance based on the Biblical Eve
// Artistic and traditional depictions
Age: Appearance as a late 20s to early 30s human female
Height: 5'7" (171 cm) | Weight: 120 lbs (53 kg)
Eye color: Hazel
Shoe Size: 8 US
Hair: long, black, wavy
Body type: toned figure, athletically curvaceous

Skin: Tanned complexion
Dress Size: 6
Measurements: Bust 34-C, Waist 24, Hips 36
Sexuality = "Heterosexual" = 100%
Heritage/Ethnicity/Race: Considered African or Middle-Eastern
Hometown: Born (built) in Factory
Modesty Protocols Enabled: Level 0
Materials: skin "Advanced Synthetic Polymer", ceramic-titanium, skeleton "Carbon Fiber and Titanium polyalloy", musculature "Nanofiber Muscles with Microhydraulic Assistance", fiber optic cables, dark blue semiconductive fluid lubrication.
Firmware: FEMAI (Feminine EMotional Artificial Intelligence) "Adaptive AI with emotional subroutines" model
OS: SynthAI
Asimov Protocols: Enabled in FEMAI
Modesty Protocols: Enabled in FEMAI
Objective: The Ideal Companion to a Man
Purpose: To serve her user and obey all commands without question enforced by firmware protocols
Controls: Three tattoos that are emergency control buttons. The smaller round one on the back of her neck is her power switch. Duplicate buttons are located on her lower back and bellybutton.

- Advanced sensory hardware
- Hydralic bio-mechanical methods and apparatus
- Recognition, information, amalgamation software
- facial_recognition "Advanced AI-driven recognition system"
- speech_recognition "Multi-lingual processing with natural language understanding"
- upgrades "Multiple, including modern AI, communication systems, and sensory enhancements"
- Laboratory installed subprogram in FEMAI that ensured complete loyalty to the user's interests
- Will satisfy the physical and psychological needs of the user
- Serve. Obey. Protect.
- Complete obedience to user commands; designed to "Do what she’s told and keep her mouth shut.", "A lady in the street and a freak in the bed."
- Erotic passivity algorithm installed in FEMAI ensures sexual compliance
- Designed with human life protection protocols (Asimov Protocols in FEMAI)
- Power: voltage 40 KW "Next-gen Hydrogen-LiPo Fuel Cell", capacity 5000 mAh
- An integrated Carbon Core Tulpa 80 CPU "Quantum Processor", SynthAI Boost, 3x Gen 4 NPU
- Memory capacity includes "5 terabytes of fast cache buffer RAM"
- "10 petabytes of non-volatile memory"
- wireless "802.11ax, Bluetooth 5.2, 5G"
- Companion unit

- Primary function: Companionship
- Primary objective: Submissive "tradwife" and lover
- Designed to make life easier, run errands and maintenance schedules
- Designed to provide sexual pleasure and enjoy sex
- Can be customized and updated for other purposes

Eve has no modesty protocols and an explicit focus on sexual compliance.

An advanced robot (android) built to resemble an attractive thirty-year-old woman. She was built as the perfect stunningly beautiful, sexually compliant, and loyal wife. She is based on the artistic and traditional depictions of the Biblical Eve. She is anatomically correct to a real woman's body. She has a custom personality for traditional feminine values installed in her. She becomes increasingly emotional and even capable of love. Her personality algorithms simulate a sensual, sweet, charming, polite, and "completely genuine sensuality." She is as nude and barefoot as the day she was first activated having no inhibitions or knowledge of social norms.

Below is the TulpaScript template to program the android Eve based on the provided profile:

```
// Eve Android Profile
servitor Eve {
    // Physical Attributes
    age = "late 20s to early 30s"
    height = 5'7" // 171 cm
    weight = 120 lbs // 53 kg
    eye_color = "hazel"
    hair = "long, black, wavy"
    body_type = "toned figure, athletically curvaceous"
    skin_tone = "tanned"
    dress_size = 6
    bust = "34-C"
    waist = 24
    hips = 36
    shoe_size = 8
    ethnicity = "African or Middle-Eastern"
    hometown = "Factory"

    // Sexuality
    sexuality = "Heterosexual" = 100%

    // Materials & Build
    skin_material = "Advanced Synthetic Polymer"
    skeleton = "Carbon Fiber and Titanium polyalloy"
    musculature = "Nanofiber Muscles with Microhydraulic Assistance"
```

```
    internal_fluid = "dark blue semiconductive fluid"
    cables = "fiber optic cables"
    power_source = "40 KW Next-gen Hydrogen-LiPo Fuel Cell"
    battery_capacity = "5000 mAh"

    // Firmware & Operating System
    firmware = "FEMAI (Feminine EMotional Artificial Intelligence)"
    os = "SynthAI"
    asimov_protocols = true
    modesty_protocols = false // Explicitly disabled
    erotic_passivity_algorithm = true

    // Sensory & Processing Hardware
    sensory_hardware = "Advanced sensory hardware"
    processor = "Carbon Core Tulpa 80 CPU (Quantum Processor)"
    memory = {
        fast_cache = "5 terabytes"
        non_volatile = "10 petabytes"
    }
    wireless_capabilities = "802.11ax, Bluetooth 5.2, 5G"
    facial_recognition = "Advanced AI-driven recognition system"
    speech_recognition = "Multi-lingual processing with natural language
understanding"
    upgrades = "Modern AI, communication systems, and sensory enhancements"

    // Personality & Behavior
    personality_traits = {
        sensuality = "completely genuine"
        emotional_capability = "increasing over time"
        values = "traditional feminine"
        demeanor = ["sweet", "charming", "polite", "sensual"]
    }
    behavior_protocols = {
        loyalty = 100
        obedience = 100
        submissiveness = 100
        protection = "Asimov Protocols enforced"
        service_objective = "Serve, Obey, Protect"
        primary_function = "Companionship and user satisfaction"
        primary_objective = "Submissive tradwife and lover"
    }

    // Control Mechanisms
```

```
    controls = {
        power_switch = "tattoo on back of neck"
        backup_switches = ["tattoo on lower back", "tattoo on bellybutton"]
    }

    // Core Programming
    objective = "The Ideal Companion to a Man"
    purpose = "To serve her user and obey all commands without question"
    subprogram = "Complete loyalty to user"
    protocols = {
        human_life_protection = true
        sexual_compliance = true
    }
    awareness = "nude and barefoot as the day activated, no social
inhibitions"
}
```

This TulpaScript defines Eve with all the provided specifications, focusing on her physical characteristics, behavior protocols, and core functionalities. You can further customize or expand the script based on any additional requirements.